Spring 13

Dynamic Adaptive Streaming over HTTP

Christopher Allen, Jean de Klerk, Natalie Kerby, and Corey Melton

May 3, 2013

Team 20:Extron

Table of Contents

Executive Summary

Our senior design project is to research the interactions between TCP and MPEG-DASH. Our sponsors are Extron Electronics - via Dr. Mike Izquierdo – Dr. Harry Perros of NCSU. Our project is based on the work of graduate students at Alpen-Adria-Universität Klagenfurt, a College of Education Studies in Austria. We were tasked with using their working model of HTTP Adaptive Streaming – a project named ITEC DASH – to create a testbed for testing adaptive streaming over HTTP.

MPEG-DASH is a dynamic adaptive protocol for streaming encoded video over the Internet. This means that the video will play at the best quality it can with little to no buffering time, given the available bandwidth. If the available bandwidth increases, the video stream will switch to a higher quality, and if the available bandwidth decreases, the video stream will switch to a lower quality in order to keep the video playing without needing to pause for buffering. A server stores a video manifest along with all the video files for each video. If the user wishes to watch a movie, then they put the location of the video's manifest file into the VLC player. The VLC player will then determine which video stream to use, and constantly monitor the buffer, input bit-rate, and the available bandwidth.

Phase one began with studying several research publications from this team and on the topic of adaptive streaming, after which we created a plan of action, compiled a list of hardware we'd need, and assembled and configured our testbed, completing phase one of our project. Our test bed includes four desktop computers, each with a specialized role to play within the network. At the center of our network is a switch that allows us to branch the network across two subnets. One of the subnets houses the HTTP Server and the Network Emulator while the other includes the Client and Traffic Generator. This setup allows us to emulate a full network environment through which to stream video. The majority of our challenges during this first phase revolved around creating a specific plan of action, building the network configuration from the ground up, and creating several scripts to maintain the network configuration via IP tables.

Phase two is the software equivalent of phase one and the majority of it included creating the testbed environment for streaming. This began with finding a decoder that worked, creating a connection from the server to the client through the network, and finally finding a workaround for our lack of working encoder. By far the most challenging aspects of this phase were the encoder and decoder. The decoder listed in the ITEC DASH project was unable to properly compile, while the decoder that we agreed upon – VLC – was unable to stream resolutions high enough to judge changes in visual fidelity. We managed to end phase two by obtaining a streaming video that had a full range of video segment qualities at a single resolution of 480p.

At the end of phase two, we had a testbed and a working streaming video using the MPEG-DASH protocol. The main goal of phase 3 was to conduct tests and collect data in order to analyze MPEG-DASH. In order to complete this goal, we had to decide on a test plan, create automation scripts to run repeated tests and collect data, and to analyze the data. We used Microsoft Excel to create graphs from our data in order to better analyze it.

Our results were very revealing, but unfortunately they were ultimately more revealing about the nature of VLC's implementation of MPEG-DASH than MPEG-DASH itself. After analyzing the data from the GET request vs. network bandwidth test, it seems that VLC has trouble 'stabilizing' the video quality at a certain range. It seems to continually jump up *and* down in bandwidth requests in an attempt to stabilize. The next two tests, GET request vs. packet loss and GET request vs. packet corruption, were completed during the last two days of our class. The most we can say from these results is that it is clear there is some interaction between TCP and MPEG-DASH, and that it appears as though this interaction is somewhat invisible until about 3% loss / corruption, but becomes increasingly evident thereafter.

Introduction

Sponsor Background

Extron Electronics is a large manufacturer of professional Audio/Video hardware and software that was established in 1983. Their products are used to create Audio-Visual presentation systems for universities, presentation centers, boardrooms, and other applications. Extron has a number of offices around the globe in order to provide full-service to their customers world wide, and their headquarters is located in Anaheim, CA.

Problem Statement

The goal of this project is to create a streaming media testbed using open source software in order to study the performance characteristics and possible interactions between the adaptive mechanism and the standard protocol stacks, as described in the recent paper "Dynamic Adaptive Streaming over HTTP Dataset", Proceedings of the Second ACM Multimedia Systems Conference (MMSys), pp 89-94 (2012). Operation of the testbed should be demonstrated by reproducing experiments described in the above paper.

A stretch goal is to explore the effect of TCP interaction on the congestion control mechanism of HTTP Adaptive.

Impact of Research

Our research will illuminate flaws and areas of interest in the cutting edge adaptive streaming over HTTP technology. Furthermore, by creating a documentation of our HTTP streaming testbed setup based entirely on open source software, we are paving the way for future research into the area and dramatically reducing the learning curve and cost to entry for this technology. Lastly, our results will provide insightful information for our sponsor, Extron Electronics, that will enable them to include MPEG-DASH support in their product line. Through MPEG-DASH products Extron would be able to provide to the market streaming video appliances that would work with a wide range of devices.

Challenges

Phase 1

Phase 1 challenges revolved around creating a plan of action and configuring the network for a two-subnet model. The MPEG-DASH format is in its infancy, which means there are many open questions and many unexplored avenues of investigation. To overcome the challenge of choosing specific questions to investigate and one avenue of attacking the problem, we spent a couple of weeks studying research on the subject and meeting with our sponsor. Eventually, we were able to narrow our focus and agree on a plan of action. The challenge of setting up the network was that of a learning curve; our resident network experts had to spend a week familiarizing themselves with subnets and IP tables before finally arriving at a stable, self-maintaining network model.

Phase 2

Our second phase involved problems of the more technical nature. This phase dealt with setting up all the various links between server and client that allowed for MPEG-DASH streaming. This was troublesome on multiple fronts: the ITEC DASH encoder was unable to compile, the HTTP server needed videos that were unable to be pulled with ftp, and the decoder refused to play video above a very low bandwidth. We began with the encoder problems: after spending several days attempting to configure and correct its flaws, we decided to grab pre-generated videos from ITEC DASH for the sake of looming deadlines. These videos, however, were ftp-locked. To work around this, one of our team members created a script that would batch pull huge amounts of files directly from the ITEC site and properly organize them on the server.

Finally, once all the pieces were working, we began to work on our last challenge: the fact that our decoder was not accepting video higher than 500kbits/s; eight times lower than what it should be requesting (in other words, terrible 240p instead of high-definition 1080p). After many hours spent attempting to isolate and solve this problem, we finally were able to obtain a streaming video that had a full range of video segment qualities at a single resolution of 480p.

Phase 3

In phase 3, our challenges revolved around testing and scripting. Our first challenge in our last phase was to identify targets of investigative interest; that is, we needed to meet with our sponsor and create a plan for the use of our testbed. Our primary concerns at first were: Which questions should we be asking, how should we answer these questions, and how do these results relate to the bigger picture of MPEG-DASH? We went about this by brainstorming ideas between ourselves and presenting a set of ideas to our sponsors, Dr. Izquierdo and Dr. Perros, at which point we narrowed down the tests to confirmation and TCP. These tests were chosen as direct influencers of MPEG-DASH, allowing us to explicitly test the inner workings of the protocol.

The next challenge we faced was to automate our testing. To properly analyze MPEG-DASH we wrote a bash script that would synchronize Wireshark's and VLC's startup. This process allowed for our tests to be easily repeated and it also minimized human error during the tests. One of our lasts challenges was to find a way to incorporate our WanEM adjustments into our testing scripts. This was solved by discovering the command line prompts on the web interface and adding the exact commands into a script that would SSH into our network emulator during the testing.

Background Research

Related Research

This new area of research has sparked lot of interest across the globe. Below are some of the papers we have studied to gain a better understanding of our topic.

- "The MPEG-DASH Standard for Multimedia Streaming Over the Internet" by Anthony Vetro, an overview of the MPEG-DASH format including a description of use, segmentation, and scope.
- "Adaptive HTTP Streaming and HTML5" by Mark Watson, a look at how Netflix implements an adaptive streaming model for tens of thousands of users.
- "Will HTTP Adaptive Streaming Become the Domain Mode of Video Delivery in Cable Networks" by Michael Adams, describing how adaptive streaming works and detailing several advantages over traditional cable network structures.
- "An Experimental Evaluation of Rate-Adaptive Algorithms in Adaptive Streaming over HTTP" by Akhshabi, Began, and Dovrolis. This paper was incredibly helpful in setting up our methodology; ours is modeled on the same general approach, despite their using Microsoft Silverlight where we use VLC. As we begin preparing for Phase 3, we will also re-reference this paper for ideas on data analysis.
- "Dynamic Adaptive Streaming over HTTP – Design Principles and Standards" by Thomas Stockhammer, a paper that detailed the advantages of HTTP streaming as well as the intricacies of 3GP-DASH, and describe the future of the MPEG-DASH project.

Work Completed by Others

For our project, we have been asked to replicate and expand on an open source experiment conducted by Lederer, Müller, and Timmerer of Austria. Their research was documented in a paper, "Dynamic Adaptive Streaming over HTTP Datasets."

Methodology

Process Flow

To begin, we broke our project into three phases listed below:

Phase 1

Constructing a multi-subnet testbed that will be able to test the interactions of MPEG-DASH and TCP.

Phase 2

Create a working video stream from Server over the Network Emulator to the Client using MPEG-DASH.

Phase 3

Analyze and investigate MPEG-DASH format over HTTP; look for interesting behavior using Network Emulator to shape traffic conditions and look for any interesting behavior in the streaming video while using the Traffic Generator to flood the network with TCP packets.

Action Steps

Below are the phases and their respective tasks.

Phase 1

- Determine hardware necessary to complete test bed
- Once accessible, create a simple testbed with a Server, Network Emulator, and Client that can be adapted to the desired test environment
    - Create routing tables to allow each element to communicate with the proper components
- Download all necessary software
- Create a document with setup instructions so that the testbed can be replicated

Phase 2

- Download all files necessary to start streaming videos.
- Configure the NetworkEmulator to a point where we can control the bandwidth, time delay, and packet loss on the connection from the server to the client
- Find and determine proper filters to apply to WireShark to capture the necessary packets.
- Create any and all, if applicable, scripts that will be used for setup and/or testing of the network.

- Create a Test (“Experiment”) Plan of what the different cases are that we will investigate with MPEG-DASH. (i.e. Drop bandwidth in intervals, simulate a network disconnection where we complete drop the bandwidth for a certain amount of time and then bring it back, etc.)
- Begin to stream video using MPEG-DASH with WireShark and WANem.
- Begin to test the Traffic Generator for TCP packet creation

Phase 3

- Run Experiments from the Test Plan
- Start analyzing the MPEG-DASH packets and video quality:
    - Keeping each of the test groups together and organized
    - Map results against each other to find R trends.
    - Start creating visuals to help with explaining the results
- Add test cases to the plan and experimentation in areas that are found to be interesting.
- If desired, add any changes to the testbed configuration

Resources Needed

There are many resources needed to complete this project and at this point in time, we have acquired all necessary resources. To help keep it organized, we have divided it into two different types of resources, hardware and software.

Hardware

- 4 desktop computers
    - Each with keyboard, mouse, and monitor
    - Ubuntu installed on 4 of them, while the Network Emulator boots into a simplified version of Knopix
    - The Server and Network Emulator each have 2 NICs, while the other two computers have only 1 NIC
- 5 Ethernet cables
- 1 5-port switch

Software

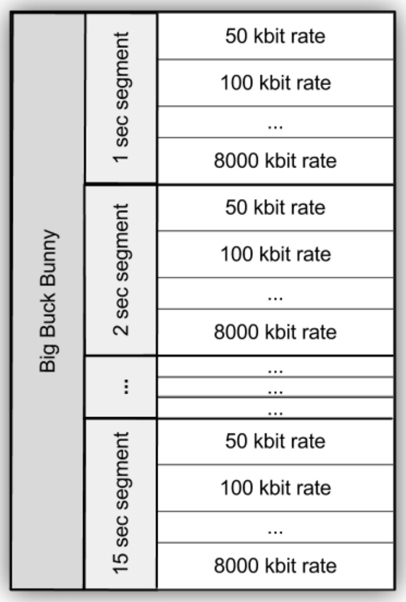
- Ubuntu 12.10
- Apache
- WireShark
- PackETH
- WANem
- VLC
- Cisco WebConnect
    - For VPN remote access
- Built in VNC client

Experiment Setup

Our testbed includes four desktop computers that act as their own part in the testbed; the HTTP Server, Network Emulator, Traffic Generator, and the Client.

HTTP Server

The HTTP Server is the only machine that is directly connected to the Internet and to the Network Emulator, which is then connected to the five-port switch; the other two machines are directly connected to the switch. This component will be used to house all of our video files and their corresponding manifests. The video files are full-length movies that are broken into different segment lengths and different bit-rates. The videos start out has full-length .mp4 files that are put through an encoder that breaks the video into multiple uniform-length segments. The encoder will also generate many copies of different bit-rate resolutions of all the segments. For example, the following table shows a basic break down of the file structure that is generated by the encoder.



Big Buck Bunny Files Organization: *This diagram helps represent how the big buck bunny file system is organized.*

Network Emulator

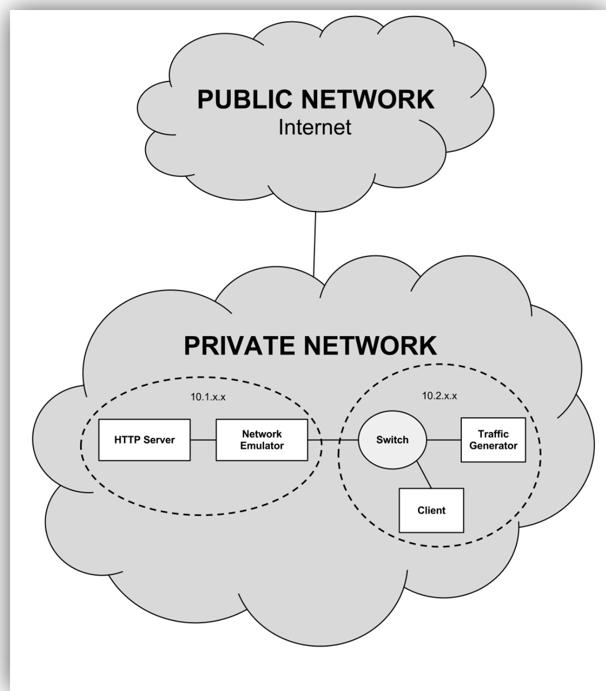
The Network Emulator connects the five-port switch to the HTTP server and acts as bottleneck for the MPEG-DASH streaming. This is a key component for our research because it allows us to simulate different bandwidths, time delays, and percentage packet loss in different sequences over time. We use a lightweight version of Knopix, which is controlled via a program called WANem, to change all of these settings.

Traffic Generator

The Traffic Generator is connected to the network via the five-port switch and will be used later in our research. Eventually it will be used to generate TCP traffic across the network using a program called PackETH.

Client

The Client is also connected to the network via the five-port switch. This component is used to view streaming MPEG-DASH video that is coming through the network. The Client has the nightly build of VLC installed so that it is able to successfully stream the MPEG-DASH video as well as display its current statistics for incoming bit rate of the video stream, which our team will capture and analyze.



Network Diagram: *This is a high level diagram of our network.*

Testing

Software/Hardware Testing

Our project’s first and second phases have no components that can be tested. The testing for this project rests heavily within the third phase but some testing is needed to ensure our testbed is functioning properly. The following tests are performed to ensure such a case.

Testbed Tests			
Test Case	**Steps**	**Expected Results**	**Actual Results**
WireShark	1. Open Wireshark 2. Select "Show the capture options..." 3. Select the interface you want to capture on. On our HTTP server it is ETH1 that goes to the private networks. 4. Double click the interface to bring up the "Edit Interface Settings" window. 5. Select "Capture Filter" scroll through the list and select the "GETs" filter. This filter was made by us, the actual filter data is "(port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420) and host 10.2.1.2". This tells it to filter GET messages from a specific host. 6. Click "OK" to close out the settings menus. 7. On the "Capture Options" menu select "Start". Wireshark will now be capturing packets.	As long as the video is running on the client, one will see a continus stream on 'GET' packets showing on the screen	
Network Setup Test (Every Machine)	1. Open a Terminal 2. Input the following commands: ping -c 1 10.1.1.1 ping -c 1 10.1.1.2 ping -c 1 10.2.1.1 ping -c 1 10.2.1.2 ping -c 1 10.2.1.3 ping -c 1 www.google.com	team20@Client:~$ ping -c 1 10.1.1.1 PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data. 64 bytes from 10.1.1.1: icmp_req=1 ttl=63 time=0.299 ms --- 10.1.1.1 ping statistics --- 1 packets transmitted, 1 received, 0% packet loss, time 0ms rtt min/avg/max/mdev = 0.299/0.299/0.299/0.000 ms	

Testbed Test: *This is a table of all tests that need to be completed to ensure that our testbed is functioning as intended.*

Performance Testing

We decided to focus our testing on three main experiments, GET requests vs. network bandwidth, packet loss vs. GET requests, and packet corruption vs. GET requests. We started our testing by creating a bash script to help automate the entire process. This bash script is setup so that it will synchronize Wireshark's and VLC's startup. This process allowed for our tests to be easily repeated and it also minimized human error during the test.

See below for a table of our tests with their respective variables:

	Independent Variables		Dependent Variables	
Tests	**Description**	**Units**	**Description**	**Units**
Validation	Bandwidth level	kbits	GET packet level requested by VLC	kbits
TCP Packet Loss	Packet loss	% packet loss	GET packet level requested by VLC	kbits
TCP Packet Corruption	Packet corruption	% packet corruption	GET packet level requested by VLC	kbits

Performance Testing*: This is a table of all experiments we ran on our testbed with their respective variables.*

The first experiment we completed is known as our validation test. This test involved mapping the GET requests from the client against the available bandwidth on the network. We gathered this data by watching the packet captures during the first 15 seconds of the video.

Our following two experiments were created after a progress meeting with Dr. Perros. It was suggested that we refocus our tests on the interactions of TCP with MPEG-DASH. Following the meetings we came up with the experiments that would analyze packet loss and packet corruption.

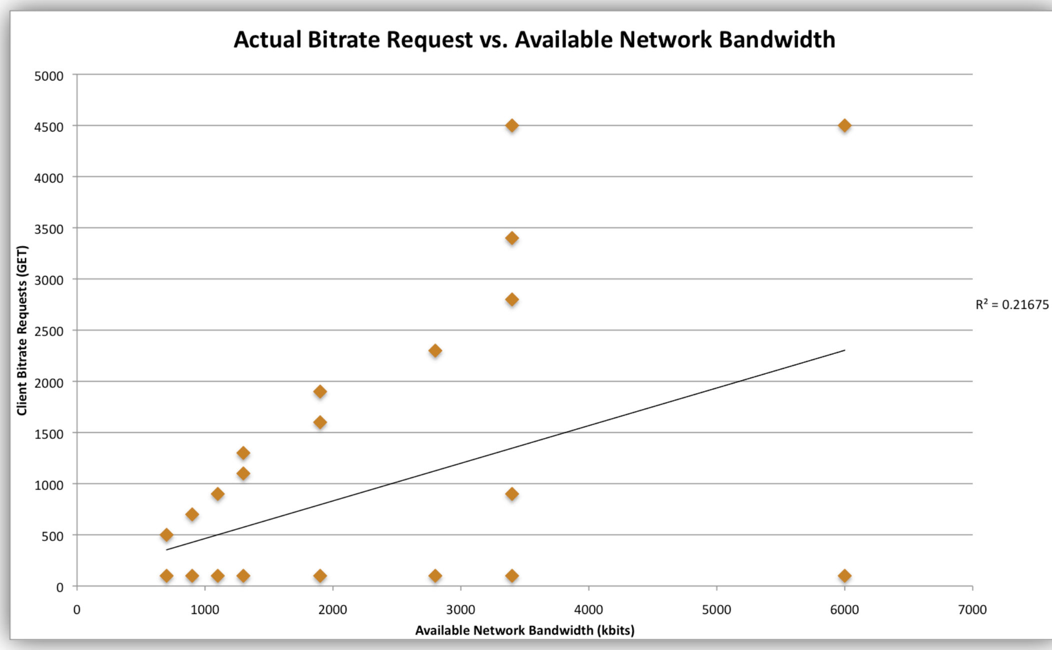
After collecting the data, we parsed the output files in excel and combined all of the separate data for each test into one dataset for graphing purposes.

Results and Discussion

Our results were very revealing, but unfortunately they were ultimately more revealing about the nature of VLC's implementation of MPEG-DASH than MPEG-DASH itself. This is in part due to the limited scope of the data we collected, and more importantly the time we spent analyzing this data due to time constraints. In what follows we will briefly discuss our results respective to each of the three tests we conducted.

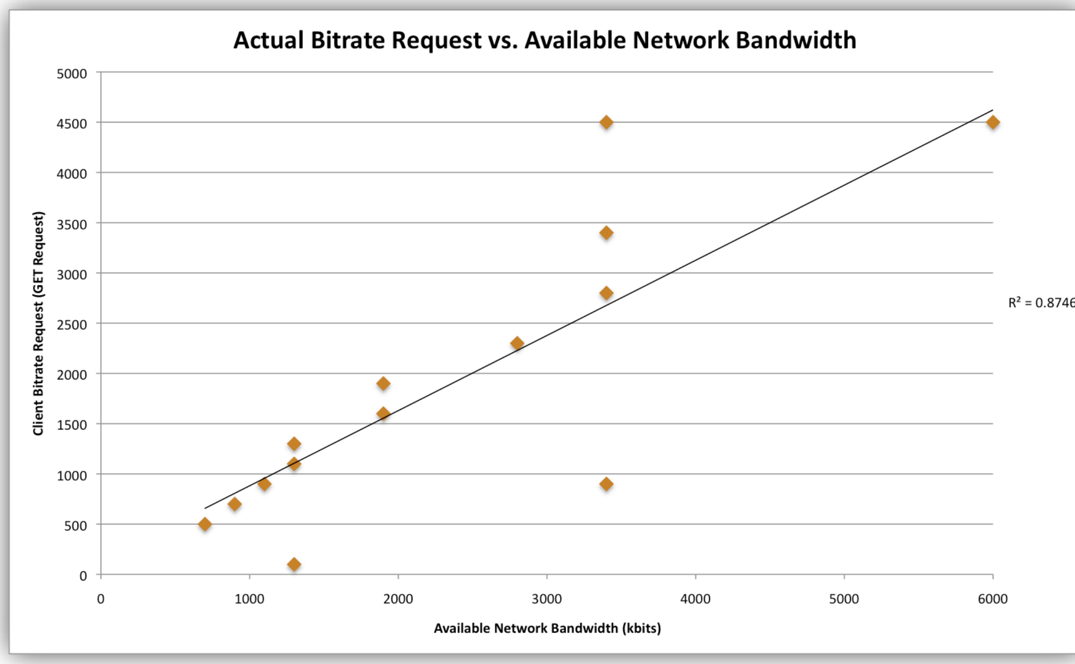
GET requests vs. network bandwidth

This test is perhaps one of the most interesting. What this test attempting to show was simply that VLC was acting as intended; that is, if our bandwidth was at 3500kbps, we expected VLC to request 3500kbit packets. See the results below:



Validation test with 'spool up': *This is a graph of our validation test with the buffer spool ups.*

It is immediately apparent that there is no clean 'one-to-one' relation between bandwidth limit and packets requested. There are two reasons for this. The first is that VLC 'spools up' at the beginning of playback; that is, it fetches packets at the 100kbit range *regardless* of available bandwidth until it fills its buffer (we set the buffer to 10 seconds, so VLC would typically fetch 5 'spool up' packets to fill its buffer and then 1-2 to refill whatever it had played in the time it took to fill the original 5). This behavior is clearly intentional and understandable; having small 'spool up' packets allows the video to start almost immediately. Let's take a look at our results without these 'spool up' packets:

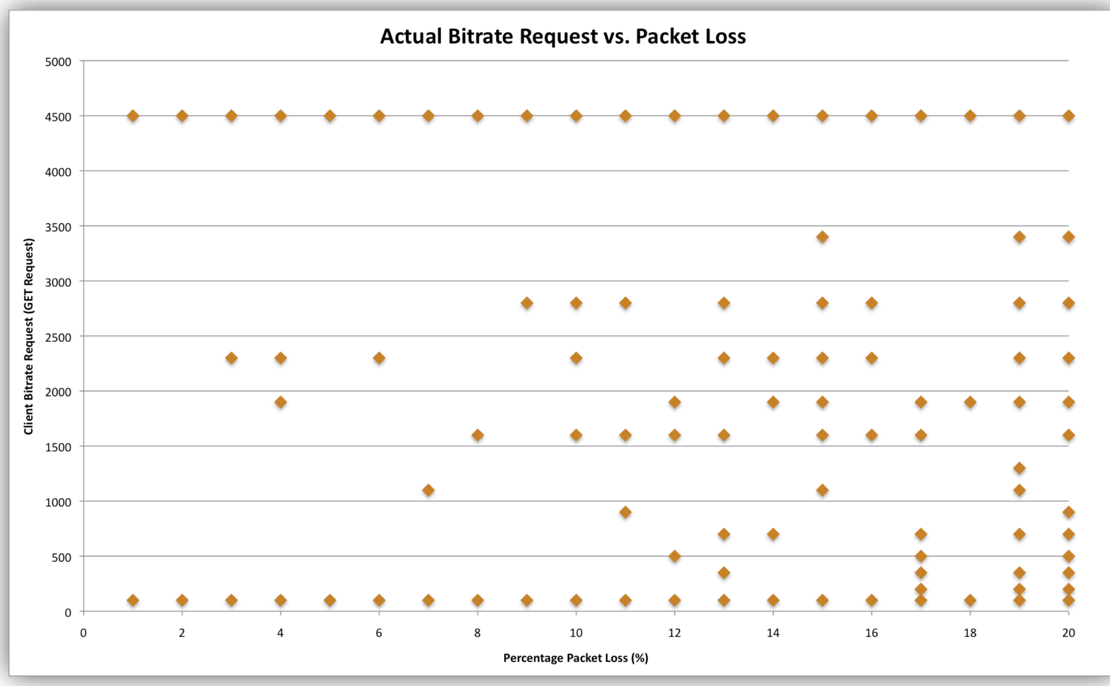


Validation test without 'spool up': *This is a graph of our validation test without the buffer spool ups.*

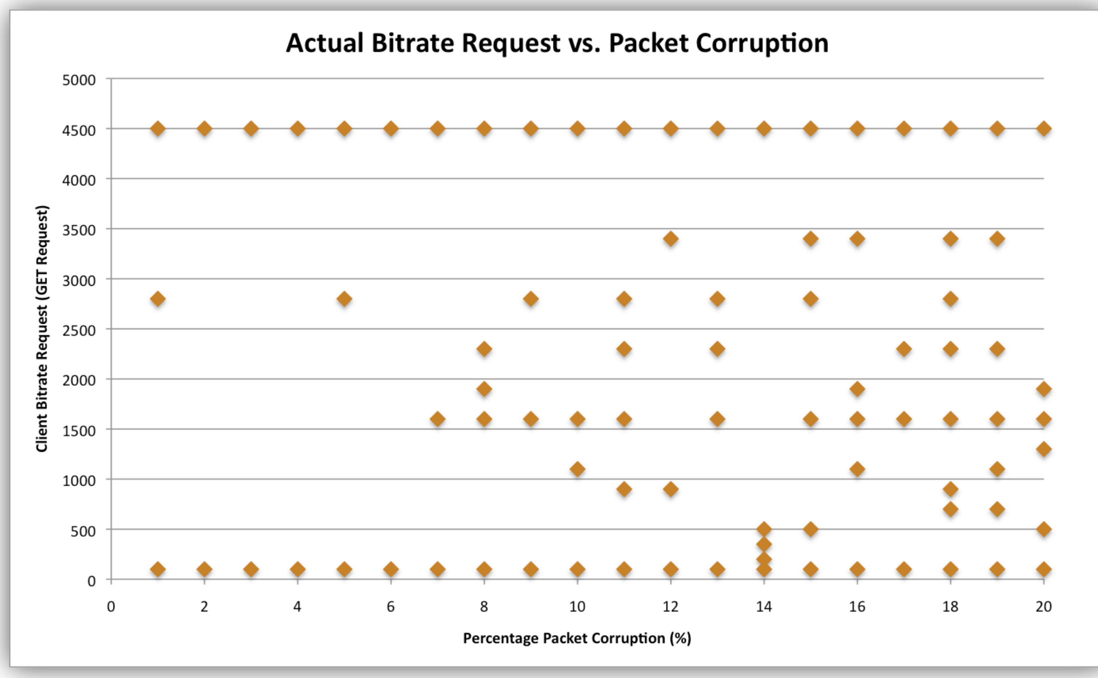
Now our attempt to find a linear fit is met with better results (R-squared of .87 as opposed to .22), however it is still messy. It seems as though VLC has trouble 'stabilizing' at a certain range and will continually jump up *and* down in bandwidth requests in an attempt to stabilize. This is another reason why VLC does not map one-to-one. Although we have no more data to continue this discussion, we did conduct some investigatory testing to see if VLC behaves differently outside of the first 15 seconds of playback. However, we were met with the same results. We suggest any future teams attempt to narrow down this problem and ascertain the reason for VLC not finding a stability point.

Packet loss vs. GET requests, Packet corruption vs. GET requests

The next two tests we conducted were aimed at seeing whether TCP interacts with MPEG-DASH. Unfortunately, these tests were conducted on the last two days of class, and though we gathered the data (see below) we were unable to conduct the significant R analysis that this data deserves. The most we can say from these results is a) that it is clear there is some interaction between TCP and MPEG-DASH, and that b) it appears as though this interaction is somewhat invisible until about 3% loss / corruption, but becomes increasingly evident thereafter. The extent to how evident it becomes is in the realm of analysis further than our own. We recommend any future teams use our extensive documentation to quickly set up a testbed and more closely analyze these interactions.



Packet loss test: *This is a graph of our packet loss data.*



Packet corruption test: *This is a graph of our packet corruption data.*

Conclusion

Our original goal was simply to create a testbed within which we could test the interactions of TCP with MPEG-DASH. We accomplished this goal and set a new goal for ourselves: begin using the testbed to conduct research on interactions between TCP and MPEG-DASH and attempt to find evidence that justified further research. In both goals we were successful, despite many challenges along the way. Our final ‘product’ is a vast amount of resources dedicated to helping streamline the process of taking a couple of computers and creating a rich network environment capable of simulating, testing, and recording every aspect of Adaptive Streaming over MPEG-DASH.

However, we hope that our true final ‘product’ is the head start to explore MPEG-DASH. Our tests and the glimpse they provide into the MPEG-DASH protocol’s workings should augment this head start with a clear and evident need for further research. Finally, our strict adherence to open-source-only software should hopefully continue opening Adaptive Streaming so that we can avoid the mistakes of our past that led to years of proprietary, non-universal video and audio players. Combined, these efforts on our part should allow for further research into Adaptive Streaming, without the worry of figuring out how to set up a working testbed.

Suggestions for Future Teams

The most important thing that we can suggest for future teams is to stay away from Virtual Machines entirely. Virtual Machines are troublesome as remote computers and hellish as networked boxes. Setting up the testbed using physical machines is somewhat complicated - there are some specific, albeit basic, routing and networking construction - but far easier than doing so on a VM. Furthermore, with Virtual Machines it is very hard to precisely control your bandwidth, given the remote nature of each machine - on a small physical network, however, you can control it to such a fine level that you can monitor each packet sent and received.

Next, we suggest using pre-generated test cases and focusing on the decoder aspect, as opposed to the encoder. The decoder is more important to set up quickly because it makes or breaks the project, whereas there are many alternatives to any given encoder (including, as mentioned, pre-generated content).

Furthermore, we suggest familiarizing yourself with both Unix and VLC (or decoder of choice) configuration settings. These tools will need to be frequently used, and a good-to-expert knowledge of both will greatly improve the project chances of success.

Lastly, we highly recommend any future teams further investigate the interactions between TCP and MPEG-DASH. There seems to be definitive evidence that some interaction exists, but without the time to fully explore this area, we were only able to catch a glimpse of the possible interactions. We hope that our efforts to streamline the process of creating a testbed and setting up a testing environment will ease the way to a point where data collection becomes the main focus.